IoT Bootcamp Supplemental Material

Brian Rashap, Ph.D.

February 18, 2024

1 Particle Troubleshooting

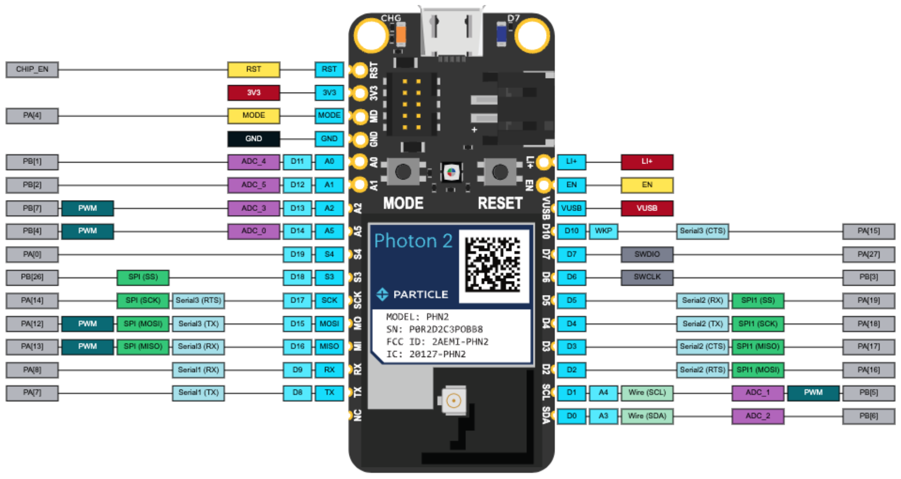


Figure 1: Photon2 Pinout

1.1 RGB LED is off

- Photon2 is not plugged in
- Short between power and ground causing an overcurrent situation To validate short:
  1. Disconnect GND pin from breadboard
  2. Remove Photon2 from breadboard and plug back in

1.2 Invalid access token

The "Invalid Access Token" error may appear in Git Bash/Terminal/PowerShell or in the VSCode terminal window (usually during Particle: Flash). The erorr most often occurs when the user is not logged into the Particle Device Cloud. To login and save an access token for interacting with your account on the Particle Device Cloud, from either Git Bash, Terminal, or PowerShell:

```
particle login
```

1.3 Not connecting to internet

- RGB LED is "breathing white" SYSTEM_MODE() is set to SEMI_AUTOMATIC or MANUAL and wifi.connect() command is absent.
- RGB LED is fast flashing green WiFi Credentials are not correct. To set WiFi credentials use:

```
particle serial wifi
```

- If resetting the WiFi credentials does not fix the issue, then often a Factory Reset is needed. Similar to putting the device in DFU-mode: hold the mode button, tap the reset, wait for the flashing to go from blinking magenta to blinking yellow to blinking white. Then release the mode button. The Photon2 will likely be in blinking blue mode, use the command above to set the WiFi credentials.

1.4 Not automatically going into DFU-Mode

If the Photon2 fails to automatically enter DFU-mode during Flash

- Manually place the Photon2 in DFU-mode. And then from Terminal/Git Bash:

```
particle flash --usb tinker
```

- If this is a recurring issue, turn on multi-treading. In the code's header, place

```
SYSTEM_THREAD(ENABLED)
```

1.5 Serial Monitor opens but no data

- Validate that the following code is in void setup():

```
Serial.begin(9600);
```

- More often then not, there is another Serial Monitor window open. Check all VSCode windows, Git Bash, Terminal, etc. Completely quit out of VSCode and start a new VSCode session if necessary.

1.6 Compile is fails without obvious errors

- Two .cpp files in /src with void setup() and/or void loop(). Remember, each .cpp file needs it own project and do not use "Save As"
- The same library added twice. For example, Adafruit_GFX gets installed with Adafruit_SSD1306, if it is also installed by itself then the compile will have issues.

1.7 Red Flash SOS

Is your device blinking red? A pattern of more than 10 red blinks is caused by the firmware crashing. The pattern is 3 short blinks, 3 long blinks, 3 short blinks (SOS pattern), followed by a number of blinks that depend on the error, then the SOS pattern again.

There are a number of other red blink codes that may be expressed after the SOS blinks:

- 1 - Hard Fault: this is most often caused by sending down over the I^2C bus without first turning on the bus with wire.begin(), bme.begin(), display.begin(), etc.
- 4 - Bus Fault (the Daniel Fault): this can be caused by trying to print a value using .printf(). If the identifier "%s" is used accidentally with an argument that isn't a char string, then the fault can be caused as the .printf() will not come across a NOLL character. Check your identifier / argument associations.

2 IoT Style Guide

A programming style guide is an opinionated guide of programming conventions, style, and best practices for a team or project. Some teams call it their coding guidelines, coding standards, or coding conventions. While these each have their own meaning in programming, they generally refer to the same thing.

In IoT, we will use the below conventions:

- Start all code with the standard comment block

```
/*
 * Project:
 * Description:
 * Author:
 * Date:
 */
```

- Provide meaningful names for variables, constants and functions. Variables and constants should be nouns and functions should be verbs.
  - The one exception to variables being meaningful nouns is that integer variables i, j, and k may be used for indexing arrays or FOR loops.
  - Constants should be in all capitals.

  ```
      const int LEDPIN = 13;
  ```

  - Variables and functions should be in camelCase[1], with the first letter lowercase: float elapseTime;

  ```
      float elapseTime;
  ```

  - Structs, ENUMs, and classes should also be in camelCase, but with the first letter capitalized.

  ```
      enum TrafficState{
        GREEN,
        YELLOW,
        RED,
        REDYELLOW;
      }
  ```

- Code should be organized as follows:
  1. header
     - #include <>
     - #include ""
     - #define
     - Data Types (e.g., structures and ENUMs)
     - Globals (constants and variables)
     - Function declarations / prototypes
     - SYSTEM_MODE()
  2. void setup()
  3. void loop()
  4. user defined functions

  NOTE: const variable should be used instead of #define.

- Avoid global variables where they are unnecessary.

[1]Camel case is the practice of writing phrases without spaces or punctuation, indicating the separation of words with a single capitalized letter.

  - Global Variables - The variables declared outside any function are called global variables. They are not limited to any function. Any function can access and modify global variables. Global variables are automatically initialized to 0 at the time of declaration. Global variables are generally written in the header-section of the code before void setup().
  - Local Variables - Local variables are declared inside of a function and only exist during that function call and are not accessible by other functions. Subsequent function calls reset the variable to its initial condition, which is 0 unless otherwise declared. Note: variables defined in void setup() or void loop() are local variables not accessible by other functions, and in the case of void loop() reinitialize during each iteration. Variables uses in void setup() and void loop() should therefore be declared as global variables.
  - Static Variables - A static variable is a local variable that is able to retain its value between different function calls. A static variable is only initialized once. If it is not initialized, then it is automatically initialized to 0.

- Opening braces should be on the same line as the conditional or function.

```
if(buttonState) {
  digitalWrite(LEDPIN, HIGH);
}
else {
  digitalWrite(LEDPIN, LOW);
}
```

- Code blocks should be indented.

```
for(i=0; i < 13; i++) {
  value = i * 3;
  if(num%2 == 0) {
    Serial.printf("The number %i is even\n",value);
    digitalWrite(LEDPIN, HIGH);
  }
  else {
    Serial.printf("The number %i is odd",value);
    digitalWrite(LEDPIN, LOW);
  }
}
```

- Use braces; avoid loops and conditionals without them.

```
// Proper use of braces:
if(buttonState) {
  digitalWrite(LEDPIN, HIGH);
}

// Avoid loops and conditions without braces. Do not do:
if(buttonState)
  digitalWrite(LEDPIN, HIGH);
//or
if(buttonState) digitalWrite(LEDPIN, HIGH);
```

- Use parentheses for clarity, especially with bit operations.

```
pixelColor = (red << 16) | (green << 8) | (blue);
//or
if((currenTime - lastTime) < timerSetting) {
  Serial.printf("Time is up \n");
}
```

- When ”printing” text and values to the Serial Monitor, OLED, or UART devices, use .printf() instead of .print() or .println()

```
Serial.printf("We use .printf() to display value = %0.2f \n",value);
Serial.println("We do not use .print() or .println()");
```

- Avoid code duplication. (i.e., appropriately use functions)

  - One of the main reasons to use a function is reusability. Once a function is defined, it can be used over and over and over again. You can invoke the same function many times in your program, which saves you work. Imagine what programming would be like if you had to teach the computer about sines every time you needed to find the sine of an angle! You'd never get your program finished!
  - Another aspect of reusability is that a single function can be used in several different (and separate) programs. When you need to write a new program, you can go back to your old programs, find the functions you need, and reuse those functions in your new program. You can also reuse functions that somebody else has written for you, such as the sine and cosine functions.

- Place a descriptive comment the line before all functions. Use comments in code as needed to clarify code logic.

```
// Get x-axis acceleration from MPU6050
float getAccX() {
  byte accelHigh, accelLow;
  int16_t accel;

  // Set the "pointer" to the 0x3B memory location
  // of the MPU and wait for data
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B); // starting with register 0x3B
  Wire.endTransmission(false); // keep active.

  // Request and then read 2 bytes
  Wire.requestFrom(MPU_ADDR, 2, true);
  accelHigh = Wire.read(); // x accel MSB
  accelLow = Wire.read(); // x accel LSB

  accel = accelHigh << 8 | accelLow;
  return accel;
}
```

3 Lab Notebook

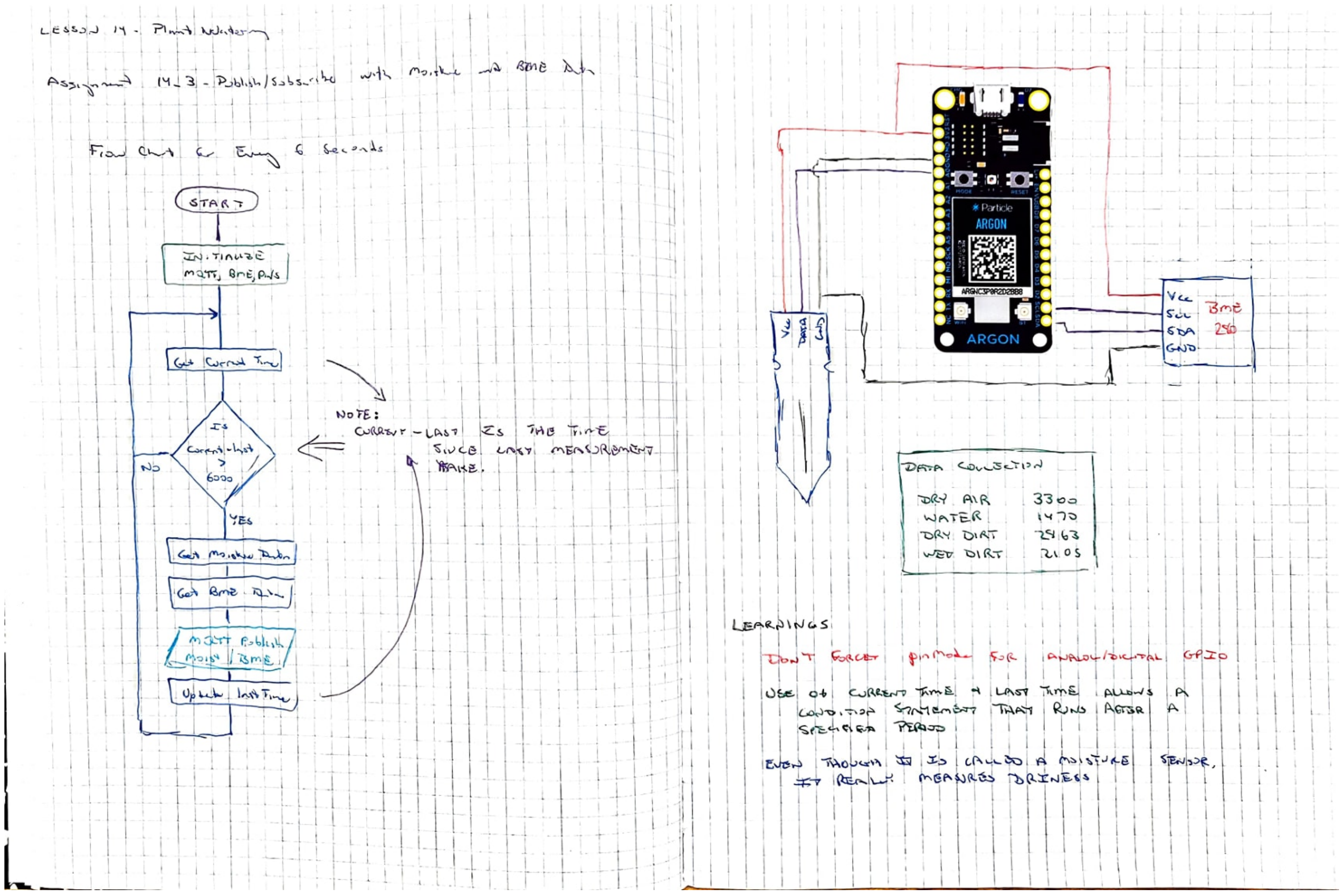
It is critically important, and often underemphasized, to keep good documentation of your projects. The above style guide assists in the readability of code. We will also keep schematics of your circuits in a program called Fritzing. The third aspect of documentation is your lab notebook. In addition to keeping notes from class lectures, each assignment will be documented in your lab notebooks.

It is expected that your assignment documentation contains the following:

- Description of assignment objective.
  - You do not need to copy the assignment verbatim; rather, you should restate the assignment in your own words.
  - Be descriptive enough so that when you come back to your notes later you can recall what you were trying to accomplish.
- Hand drawn schematic of your circuit layout
  - This is an opportunity to think about the proper connections and interface types between the components and your microcontroller.
  - Be sure to include the proper current limiting resistors, pull-down or pull-up circuits, and noise filters [2]
  - The schematic should be detailed enough that you can create your fritizing diagrams and/or wire your circuit directly.
- Flow charts and/or outlines of your code logic.

[2]All of these will be introduced as we progress through the class lessons

- Flow charts are written in english[3], not in C++ code.
- Flow charts need not outline the entire code, but rather the conditions and flow control features that are integral to the assignment. For example, the flow chart in Figure 1 outlines how to publish data to the cloud at a set interval without using a delay.
- Add comments (notes) to your flow chart to assist with readability and highlight important features, as necessary.
- In addition to flow charts, use your lab notebook to outline any finite state machines and state transitions. An example is the traffic light states seen in Figure 2 and the flow chart below it showing the logic for state transitions.



LESSON 14 - Plant Watering
Assignment 14_3 - Publish/subscribe with moisture and BME Data
Flow Chart for Every 6 Seconds
START
INITIALIZE MQTT, BME, Pins
Get Current Time
Is Current-Last > 6000
NO
YES
Get Moisture Data
Get BME Data
MQTT Publish Moist / BME
Update Last Time
NOTE: CURRENT - LAST IS THE TIME SINCE LAST MEASUREMENT MADE.
Vcc
Data
Gnd
ARGON
Vcc
SCL
SDA
GND
BME 280

DATA COLLECTION

DRY AIR	3300
WATER	1470
DRY DIRT	2463
WET DIRT	2105

LEARNINGS

DON'T FORGET pinMode FOR ANALOG/DIGITAL GPIO

USE OF CURRENT TIME & LAST TIME ALLOWS A CONDITION STATEMENT THAT RUNS AFTER A SPECIFIED PERIOD

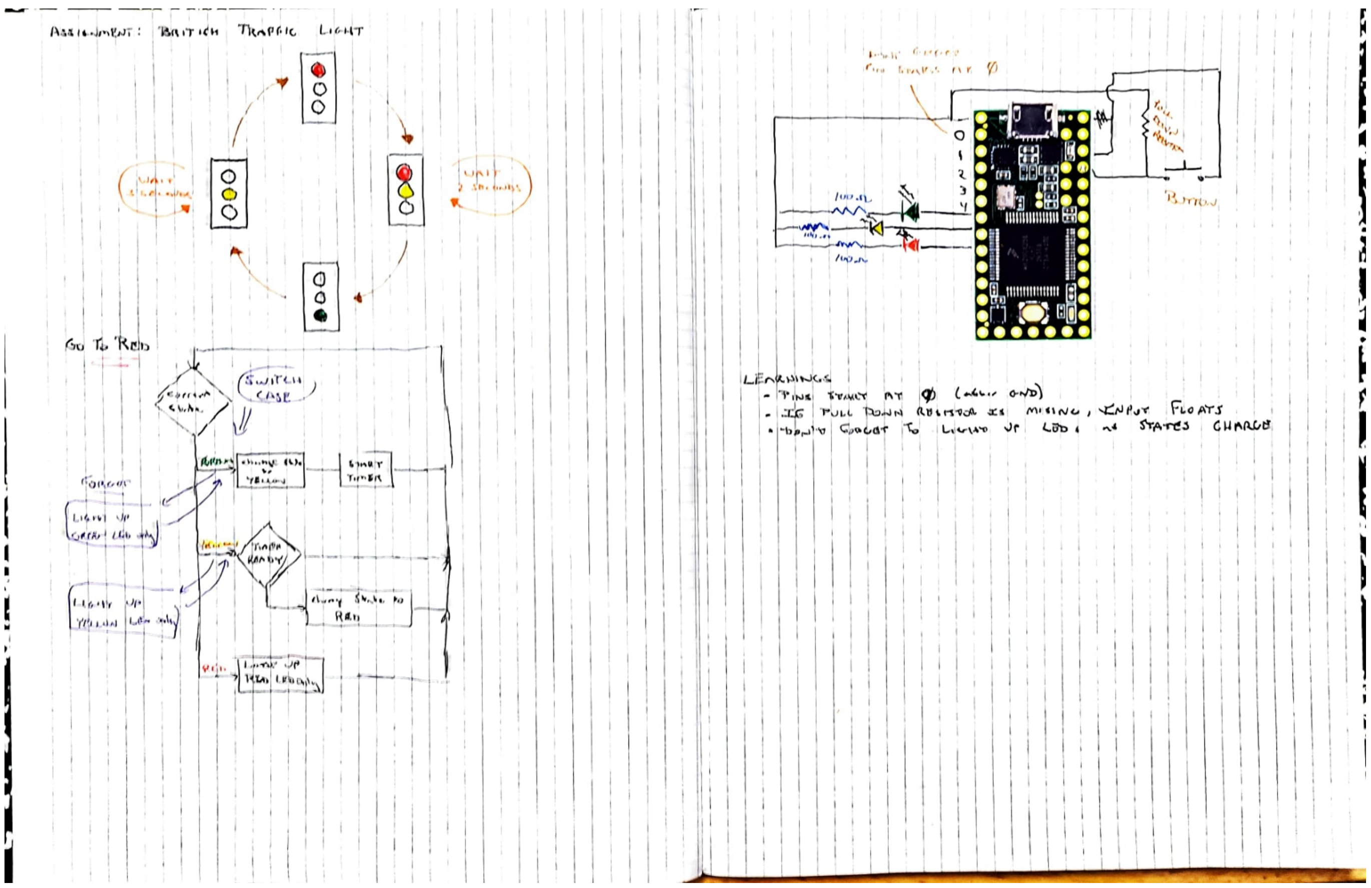
EVEN THOUGH IT IS CALLED A MOISTURE SENSOR, IT REALLY MEASURES DRINESS

Figure 2: Lab Notebook example: Soil Moisture

- Any results obtained from the assignment.
  - Most assignments will not explicitly ask for documenting results. However, where appropriate you notebook to document readings obtained by your IoT device (i.e., the box on the right hand side of Figure 1).
- A lessons learned conclusion where you document learnings that you obtain from debugging your code, circuit layout, etc.
  - At the end of each assignment, reflect on how the assignment went, what parts frustrated you, what parts gave you a sense of accomplishment.
  - Document any deeper understandings of code or circuits that you obtained beyond the preceding lecture.
  - If there are systemic errors you are making (e.g., placed a ";" at the end of the line on a condition statement, document these and the remedy for future reference.[4]

[3]English, or the human language of your choosing

[4]You should use the last page of your notebook to keep a running list of "mistakes not to make."



LEARNINGS
- PINS START AT Ø (AFTER GND)
- IF PULL DOWN RESISTOR IS MISSING, INPUT FLOATS
- DON'T FORGET TO LIGHT UP LEDs AS STATES CHANGE

Figure 3: Lab Notebook example: Soil Moisture